Core Audio Technologies

Session 502

Core Audio Technologies

Craig Keithley
USB and FireWire Technology Evangelist

Topics Covered

- AudioConverter
- 3D Mixer
- Multi-Channel Reverb
- AUGraph
- MusicDevice (DLS Synth)
- AUMIDIController
- AudioUnit View Components

Using the Audio Converter

Jeff Moore
Core Audio Engineering

Introduction

- The AudioConverter is an API for transforming audio data from one format to another
- There are converters for doing rate conversion, integer to floating point and floating point to integer conversion, interleaving and deinterleaving, and encoding and decoding

Audio Formats

- The AudioStreamBasicDescription structure provides the basic encapsulation for describing formats
- The canonical format is 32-bit floating point
- Both interleaved and non-interleaved data is supported

Encoded Audio Formats

- Encoded formats may require out of band configuration information
- Encoded formats are usually dealt with in terms of whole packets of frames of data
- In variable bit rate encoded formats, the size in bytes of each packet of data will vary
- Many encoded formats also require an external description of the packetisation

Creation and Configuration

- Specifying the basic transformation

```
AudioStreamBasicDescription theInputFormat;
AudioStreamBasicDescription theOutputFormat;
AudioConverterRef theAudioConverter = NULL;

theError = AudioConverterNew(
                        &theInputFormat,
                        &theOutputFormat,
                        &theAudioConverter);
```

Creation and Configuration

- Supplying the additional configuration information

```
void* theMagicCookie;
UInt32 theMagicCookieSize;
GetInputFormatMagicCookie(theMagicCookie,
                          theMagicCookieSize);

theError = AudioConverterSetProperty(
    theAudioConverter,
    kAudioConverterPropertyInputCodecParameters,
    theMagicCookieSize, theMagicCookie);
```

Creation and Configuration

- Changing the rate conversion algorithm

```
theConverterAlgorithm =
    kAudioUnitSRCAlgorithm_Polyphase;

theError = AudioConverterSetProperty(
    theAudioConverter,
    kAudioConverterSampleRateConverterAlgorithm,
    sizeof(OSType), &theConverterAlgorithm);
```

Transforming Data

- Simple data and simple data flow
  - Only available for transformations where the ratio between input data and output data is constant and an integer
  - Use AudioConverterConvertBuffer

Transforming Data

- Simple data and simple data flow

```
while(GetNextInputBuffer(
        &theInputData, &theInputDataSize)) {
    GetNextOutputBuffer(
        &theOutputData, &theOutputDataSize);

    theError = AudioConverterConvertBuffer(
        theAudioConverter, theInputDataSize,
        theInputData, &theOutputDataSize,
        theOutputData);
}
```

Transforming Data

- Simple data and complex data flow
  - Use when the input data handling needs more control
  - Use AudioConverterFillBuffer which takes its input from a call back

Transforming Data

- Simple data and complex data flow

```
while(GetNextOutputBuffer (&theOutputData,
                &theOutputDataSize)) {

    theError = AudioConverterFillBuffer(
        theAudioConverter,
        MyAudioConverterInputDataProc,
        NULL, & theOutputDataSize, theOutputData);
}
```

Transforming Data

- Complex data and complex data flow
  - Use when the transformation itself requires more information than the other methods provide
    - Interleaving/deinterleaving
    - Encoding/decoding
  - Use AudioConverterFillComplexBuffer

Transforming Data

- Complex data and complex data flow

```
while(GetNextOutputBufferList(&theOutputDataList,
                &theNumberPackets)) {

    theError = AudioConverterFillComplexBuffer(
        theAudioConverter,
        MyAudioConverterComplexInputDataProc,
        NULL, &theNumberPackets, theOutputDataList,
        NULL);
}
```

AUConverter AudioUnit

- Wraps up the AudioConverter in an AudioUnit for easy integration

AudioConverter Plug-Ins

- Audio Codec components
  - Covered in more detail in Session 508
- SDK available now
  - Audio Codec APIs
  - Audio Codec base classes
  - Audio Converter will not use them yet!

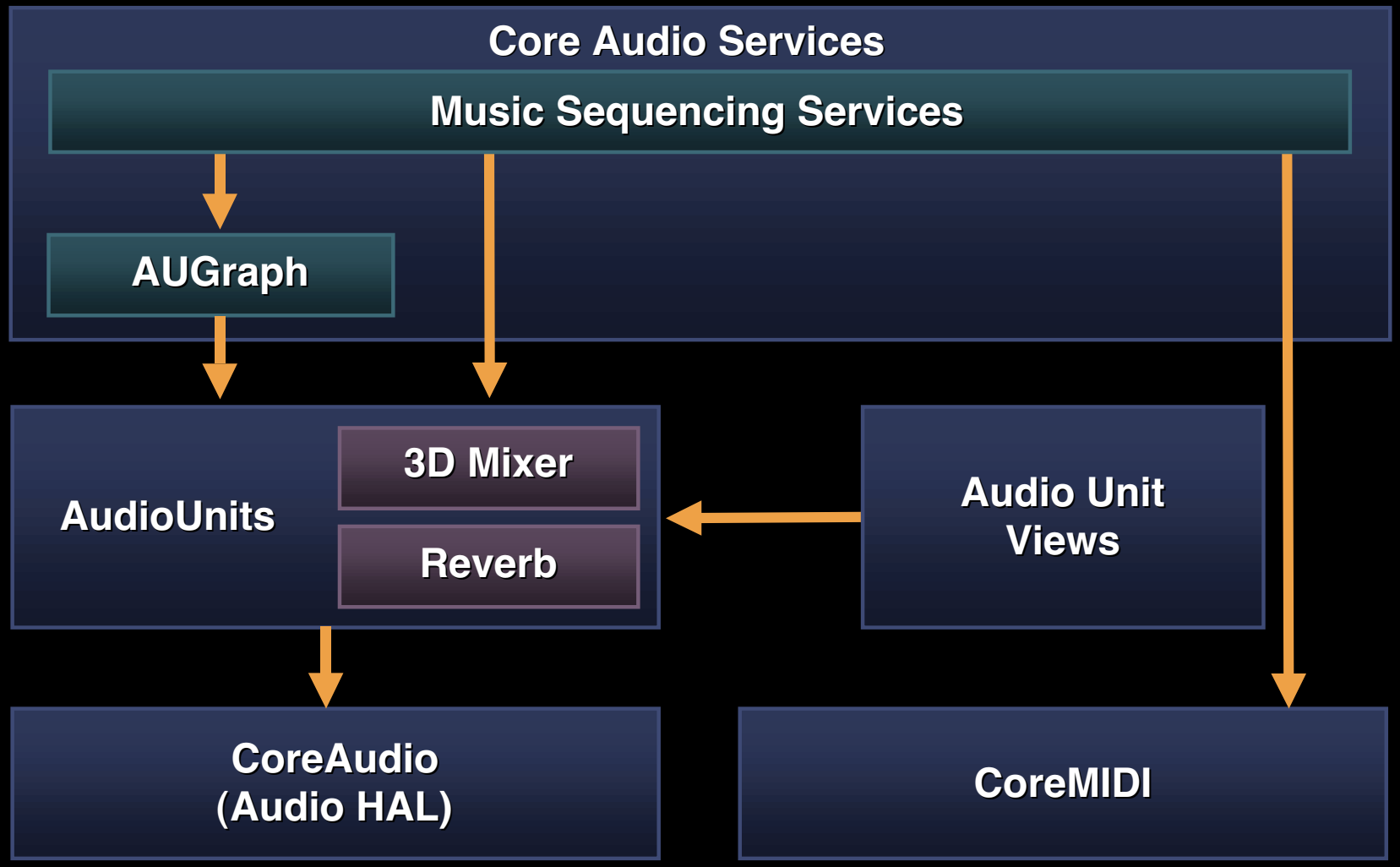
Audio Spatialization

Chris Rogers
Core Audio Engineering

Introduction

- 3D audio conceptual overview
- Positioning sounds using the 3D mixer
- Applications using the reverb AudioUnit

Architectural Overview



3D Audio Structural Overview

Sources **Encoding** **Reproduction** **Perception**

Recording Technique

Spatialization Processing

Basic Psychoacoustics of Sound Localization

- ITD (Inter-aural time difference)
- IID (Inter-aural intensity difference)
- Spherical Head Model
- HRTF
- Distance Cues

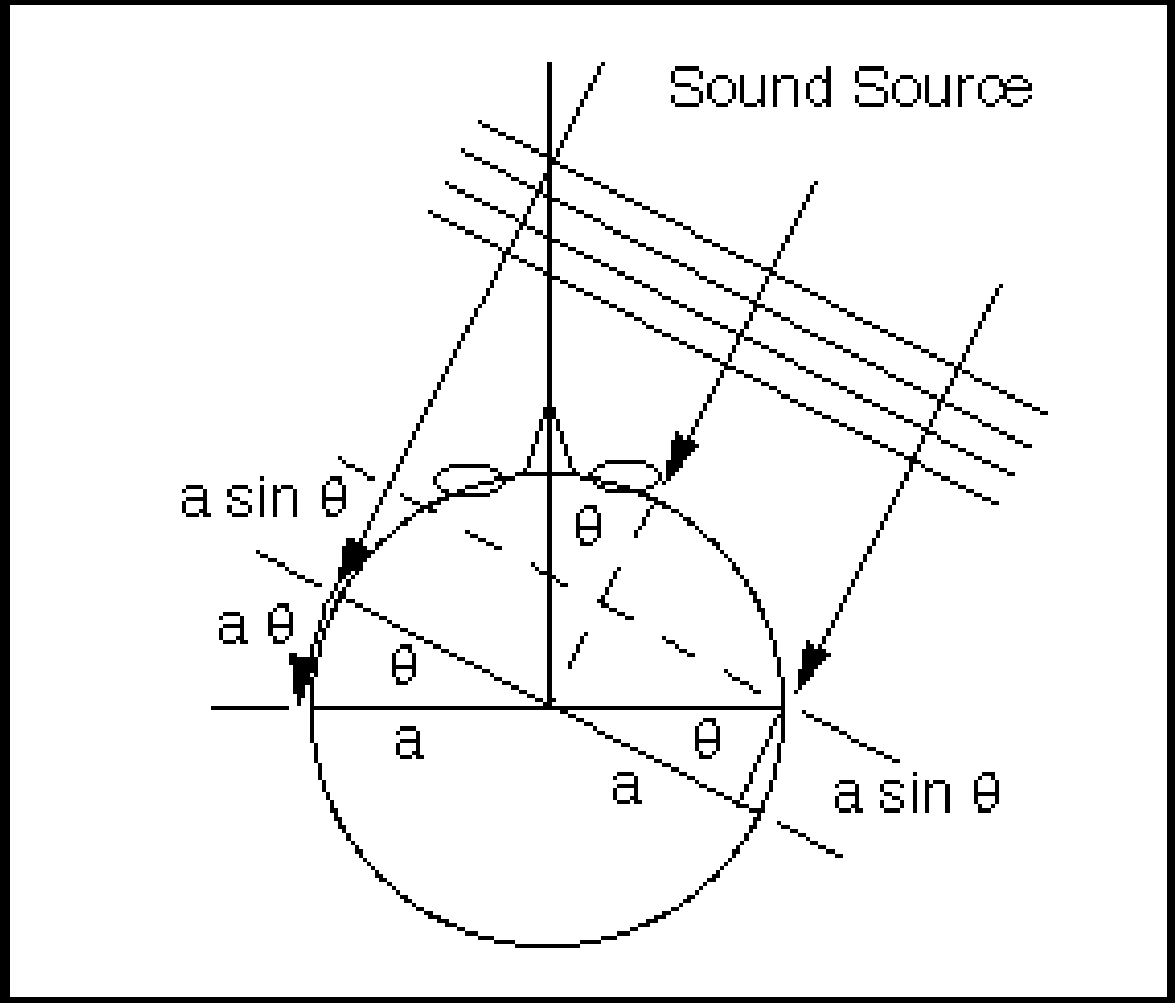
ITD

- Time difference of sound arrival to each ear
- The ear localizes sounds below 1000Hz using this method

IID

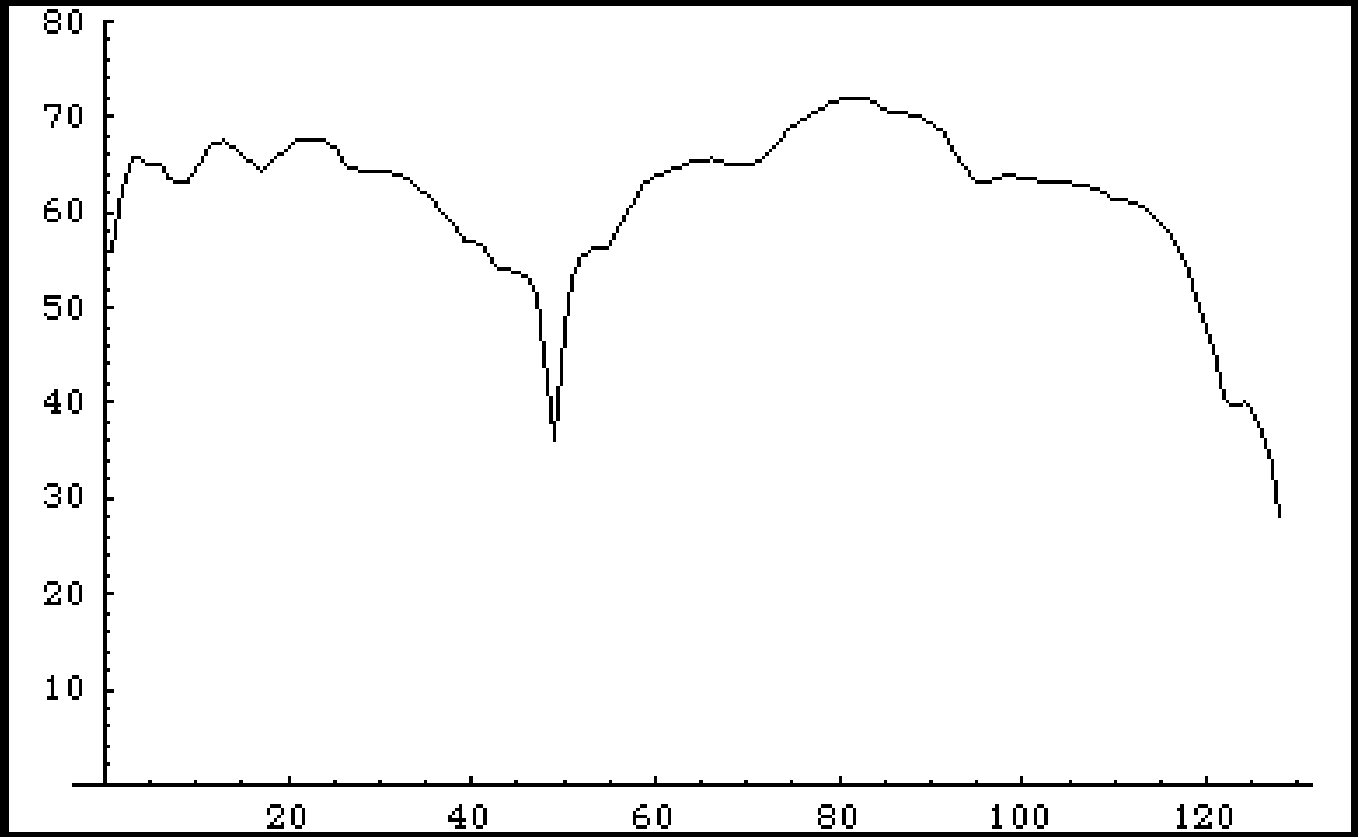
- Frequency-dependent intensity difference between ears
- The ears, head, and body act as directional filter
- We localize sounds above 1000Hz using this method

Spherical Head Model



HRTF

(Head-Related Transfer Function)



Distance Cues

- Ratio of direct to reverberant sound
- Doppler shift
- High frequency drop-off (air absorption)
- Motion parallax (close objects span large angles as they move small distances)
- IID can be exaggerated for sources very near one ear
- Diffusion

Demo

Doppler Shift

Intrinsic vs. Synthetic Sources

- "Intrinsic" recorded source
  - Microphones encode intrinsic spatial information for recording
  - Little/no further processing necessary to hear
- "Synthetic" or mono/dry recorded source
  - Must be processed to impart a spatial quality
    - 3D mixer can localize in space
    - Reverb can add ambience

Recording Techniques for Intrinsic Sources

- Binaural recording
- SoundField microphone, B-format Ambisonics
- Others: Coincident pairs, distantly-spaced omni-directional microphones, etc.

Binaural Recording

- Uses principles of HRTF/ITD/IID
- Dummy head with microphone in ears
- Personally molded ears
- Playback on headphones or crosstalk-cancelled speakers

SoundField Microphone

- Ambisonics B-format (WXYZ)
- Captures 3D soundfield
- Uses omni and three figure-eight elements to encode four channels

Ambisonics

- Recording and panning technique
- Aims to recreate the sound-field using all of the available speakers
- May work with arbitrary number of speakers
- Does not require a “sweet spot” for listening
- Sound does not collapse into a single speaker

Ambisonics Encoding

- Four channel encoding WXYZ (B-format)
  - W is omni-directional component
  - X is front-back component
  - Y is left-right component
  - Z is up-down component
- Equations exist to spatialize a “synthetic” sound into WXYZ

3D Mixer AudioUnit

- Localizes synthetic/dry/mono sources
- N mono (or stereo) input busses
- One output bus whose stream format depends on the playback configuration
- Optional built-in reverb

Parametric Source Control

- Each source responds to these parameters
  - Azimuth angle (degrees)
  - Elevation angle (degrees)
  - Distance (meters)
  - Gain

Setting Up the 3D Mixer AudioUnit

- Configurable through properties and parameters
  - Global playback configuration (default is stereo speakers)
  - Rendering algorithm settable per channel (default is Ambisonics)
  - Optional doppler shift per channel

Playback Configurations

- Mono
- Stereo headphones
- Stereo speakers
- Quad
- 5.1
- Others possible in the future (7.2, n-channel, etc.)

Spatialization Rendering Techniques

- Stereo
  - Equal power panning
  - Spherical head model
  - HRTF
- N-channel
  - Ambisonics soundfield
  - Vector-based panning

Environmental Ambience/ Acoustic Space Modeling

- Matrix Reverb Audio Unit
  - Room size
  - Density
  - Brightness, etc.

Reverb Applications

- Stereo reverb
- Surround reverb
- Stereo reverb only in rear surround speakers
- Apply reverb to mono source, then localize

Surround Reverb

- Property configures output to 5 channels
- Each of the five speakers will have separate reverberant stream
- Creates enveloping reverberation for 5.1 production/authoring

Demo

Surround Reverb

Demo

Directionalized Reverb

Other Audio Toolbox APIs

William Stewart
Core Audio Engineering

Music Services

- MusicSequence
  - Contains n tracks of MusicEvents
  - Create from and Save to MIDI files
  - Can play to any AudioUnit or MIDI Endpoint
- MusicPlayer
  - Used to play back a sequence

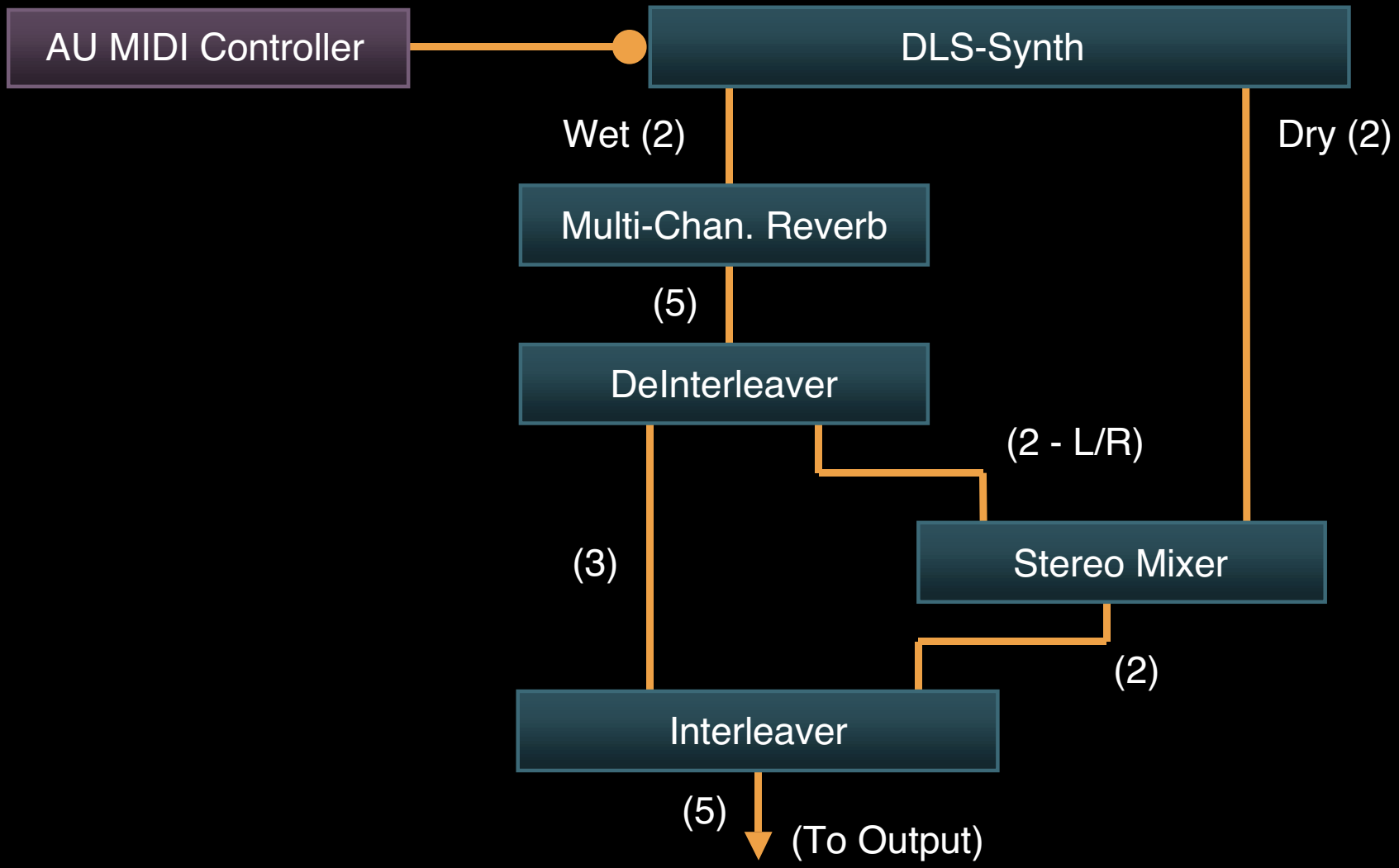
Multi-Verb Synth

- Starts with a DLSMusicDevice
  - Turn built-in reverb off
  - Take wet and dry outputs for separate processing
- Wet output through Reverb
  - 5 channel output
  - Mix first 2 channels with dry output of synth
- Interleave back to 5 channel output

AUMIDIController

- Receives MIDI events and translates them to:
  - AudioUnit Parameter calls
  - MusicDevice “SendMIDI” calls
- MIDI Events received by:
  - Connecting a MIDI source
    - Creates a MIDIInputPort for itself
  - Publishing a MIDIEndpoint (virtual destination)

Signal Flow of MVS Graph



Demo

“Multi-Verb Synth”

Roadmap

FF014 Audio and MIDI Feedback Forum Question and Answer Forum	Room J **Fri., 3:30pm**
507 Audio and MIDI: Using the Audio HAL and Core MIDI services	Room J **Wed., 3:30pm**
508 Audio Units and Audio Codecs: Writing Audio Units and Audio Codecs	Room J **Wed., 5:00pm**
607 QuickTime and MPEG-4: Short overview of AAC	Room A2 **Fri., 3:30pm**

Who to Contact

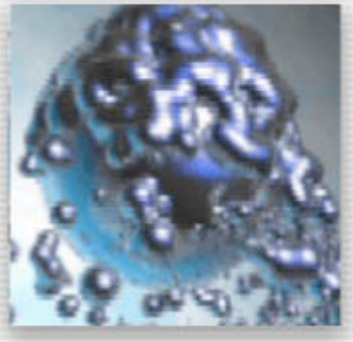
Craig Keithley
USB and FireWire Technology Evangelist
keithley@apple.com

http://developer.apple.com/wwdc2002/urls.html

For More Information

- Core Audio Developer Services
- Mailing List—Core Audio API
  - **http://lists.apple.com/**
- SDKs
  - **http://developer.apple.com/audio/**

Q&A



Craig Keithley
USB and FireWire Technology Evangelist
keithley@apple.com

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002